The Android Input Architecture

The journey of a thousand function calls starts with an Interrupt

Jonathan Levin

Technologeeks.com (@Technologeeks)

NewAndroidBook.com

About This Talk

- Discusses the Android Input Stack, in depth
- Follows flow of input (up to App, sans IME)
  - Avoids code as much as possible
- Demonstrates a few handy input tools
- Essentially an excerpt from the Book.

The Book

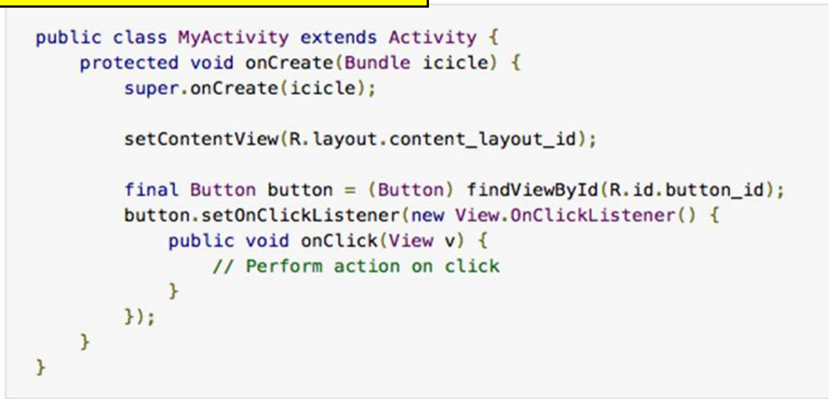
- “Android Internals:: A Confectioner’s Cookbook”
- Unofficial parallel to “Mac OS X and iOS Internals”
  - (which, btw, is coming out in a 2nd Edition for iOS 9/OS X 10.11!)
- Volume I released earlier this year
  - Already updated for Android M PR1-2!
- Volume II to be released soon
  - As soon as Google stabilizes M
- http://www.NewAndroidBook.com/
  - FAQ, TOC and plenty of bonus materials

User Apps

What do we know about input?

The activity gets the input
as an event, via the target view's
onXXX event callback

Activity gets input as part of a specified event callback



```
public class MyActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
            }
        });
    }
}
```

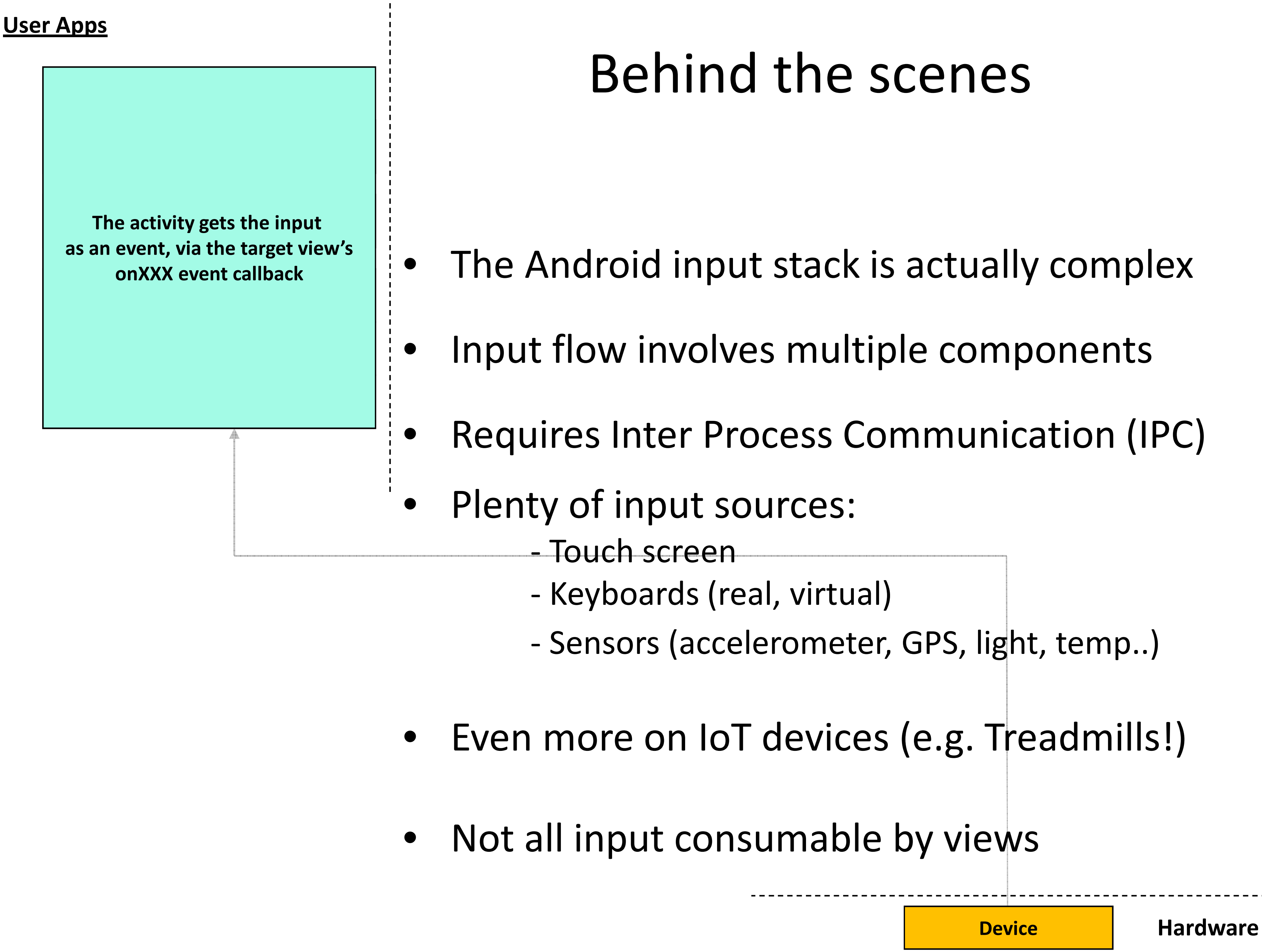
Physical events (e.g. touches, clicks, swipe, etc) occur at the device hardware level

Device

Hardware

Behind the scenes

- The Android input stack is actually complex
- Input flow involves multiple components
- Requires Inter Process Communication (IPC)
- Plenty of input sources:
  - Touch screen
  - Keyboards (real, virtual)
  - Sensors (accelerometer, GPS, light, temp..)
- Even more on IoT devices (e.g. Treadmills!)
- Not all input consumable by views



User Apps

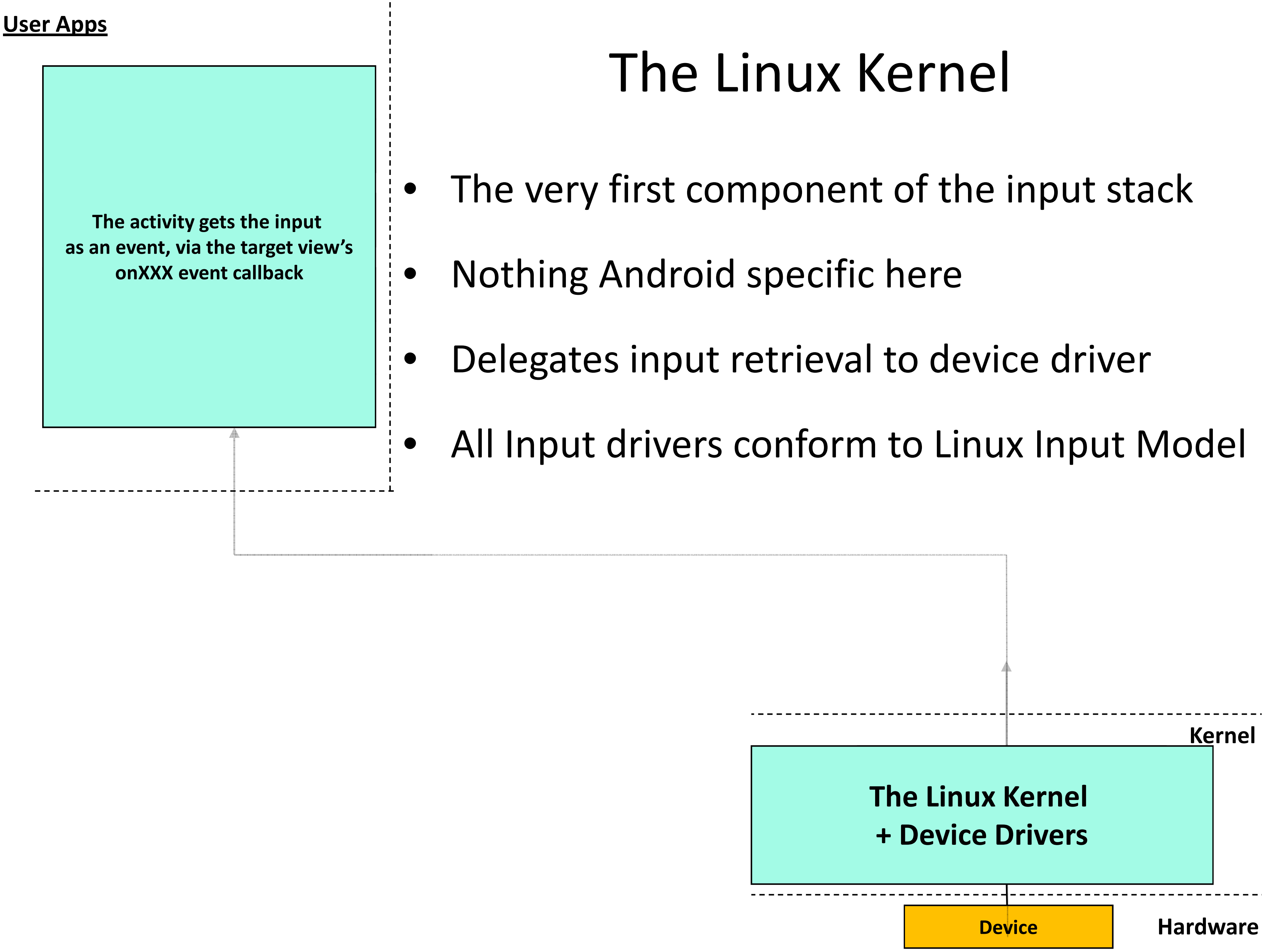
The activity gets the input as an event, via the target view's onXXX event callback

Device

Hardware

The Linux Kernel

- The very first component of the input stack
- Nothing Android specific here
- Delegates input retrieval to device driver
- All Input drivers conform to Linux Input Model



User Apps

The activity gets the input as an event, via the target view's onXXX event callback

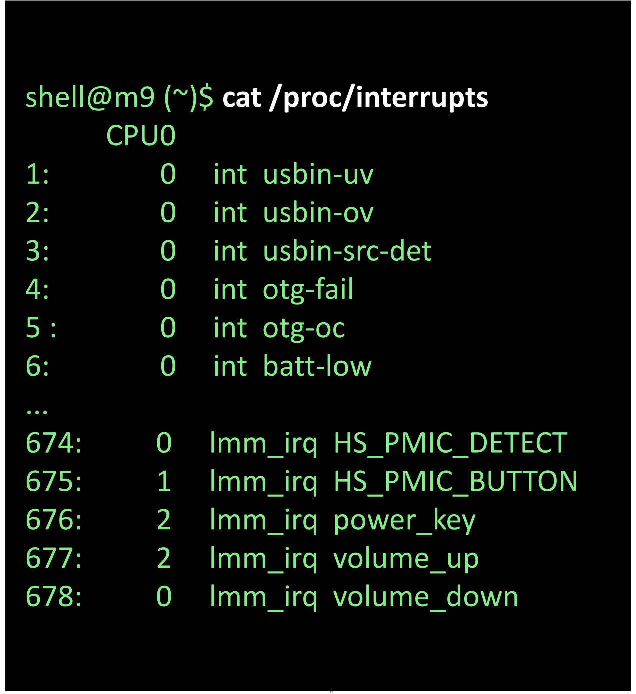
Kernel

The Linux Kernel + Device Drivers

Device

Hardware

The Linux Input Model

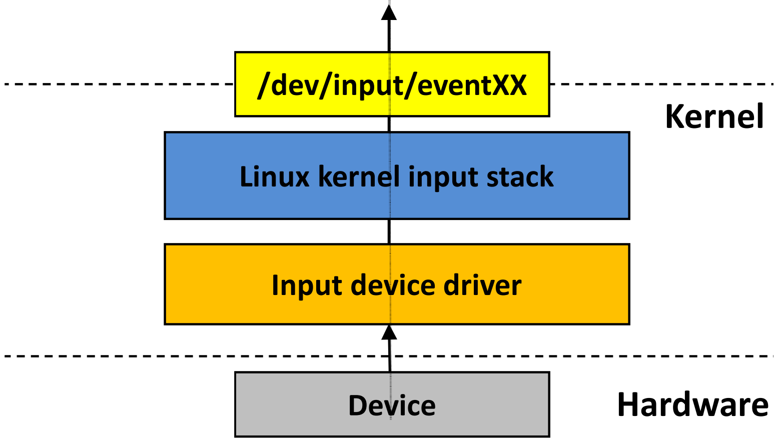


```
shell@m9 (~)$ cat /proc/interrupts
         CPU0
1:            0    int  usbin-uv
2:            0    int  usbin-ov
3:            0    int  usbin-src-det
4:            0    int  otg-fail
5 :           0    int  otg-oc
6:            0    int  batt-low
...
674:          0    lmm_irq  HS_PMIC_DETECT
675:          1    lmm_irq  HS_PMIC_BUTTON
676:          2    lmm_irq  power_key
677:          2    lmm_irq  volume_up
678:          0    lmm_irq  volume_down
```

- interrupt statistics in /proc/interrupts

(nice bonus: # of active CPUs)

- Drivers claim interrupt (request_irq)
- Driver callback invoked by kernel

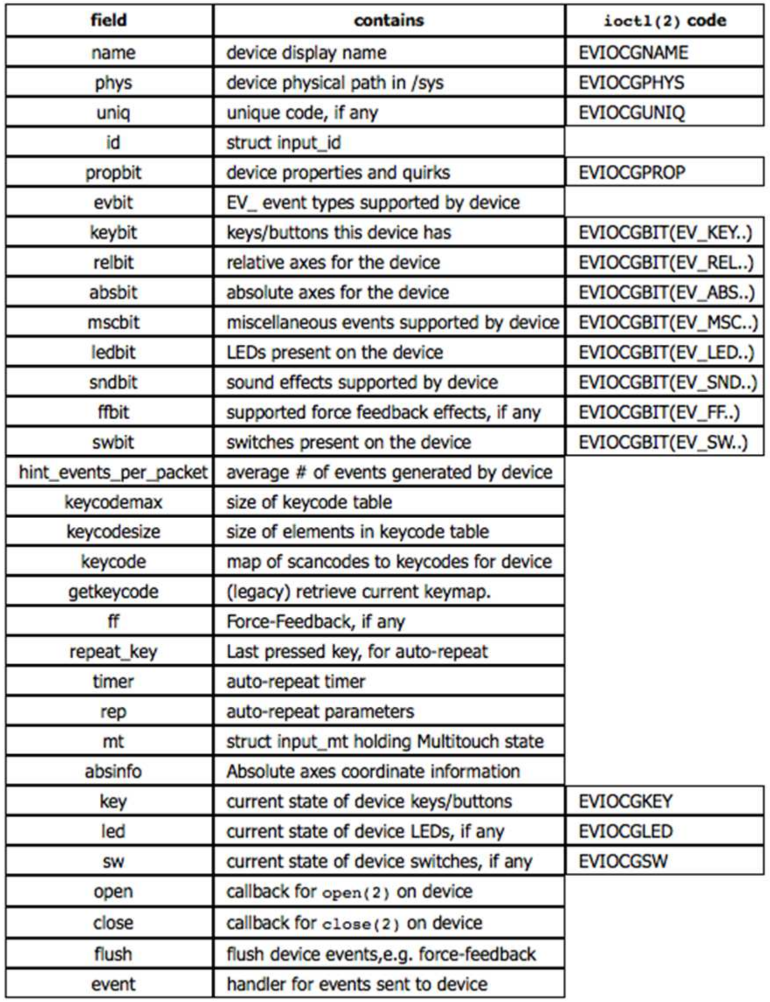


CPU responds to interrupts, calls kernel, to dispatch to device driver

All Input starts with some type of interrupt, generated by device

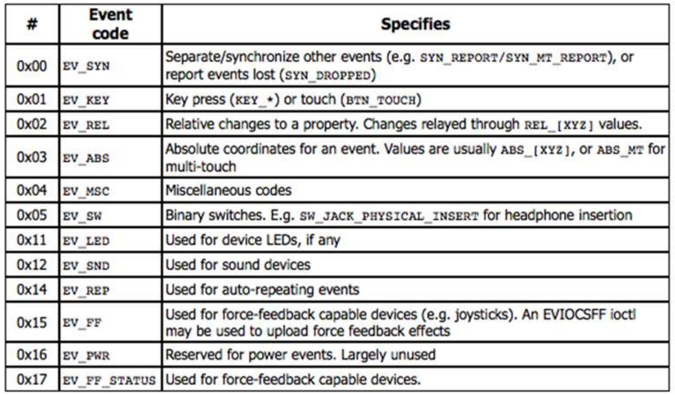
The Linux Input Model

- Driver registers input_device
- Device file created: `/dev/input/eventXX`
- Driver reports event as an event record

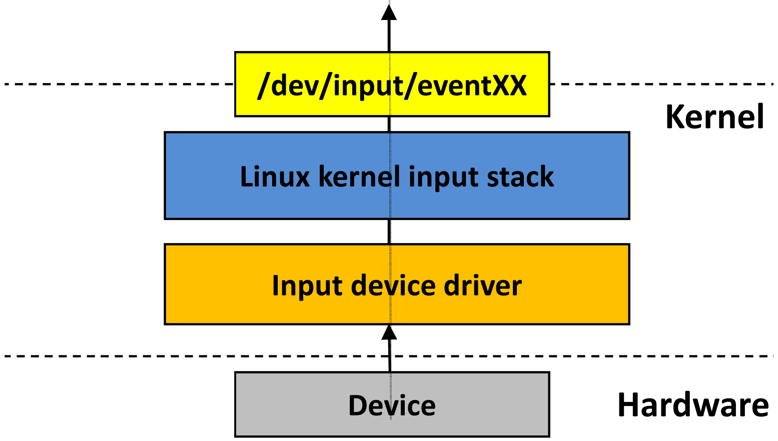


field	contains	ioctl(2) code
name	device display name	EVIOCGNAME
phys	device physical path in /sys	EVIOCGPHYS
uniq	unique code, if any	EVIOCGUNIQ
id	struct input_id	
propbit	device properties and quirks	EVIOCGPROP
evbit	EV_ event types supported by device	
keybit	keys/buttons this device has	EVIOCGBIT(EV_KEY..)
relbit	relative axes for the device	EVIOCGBIT(EV_REL..)
absbit	absolute axes for the device	EVIOCGBIT(EV_ABS..)
mscbit	miscellaneous events supported by device	EVIOCGBIT(EV_MSC..)
ledbit	LEDs present on the device	EVIOCGBIT(EV_LED..)
sndbit	sound effects supported by device	EVIOCGBIT(EV_SND..)
ffbit	supported force feedback effects, if any	EVIOCGBIT(EV_FF..)
swbit	switches present on the device	EVIOCGBIT(EV_SW..)
hint_events_per_packet	average # of events generated by device	
keycodemax	size of keycode table	
keycodesize	size of elements in keycode table	
keycode	map of scancodes to keycodes for device	
getkeycode	(legacy) retrieve current keymap.	
ff	Force-Feedback, if any	
repeat_key	Last pressed key, for auto-repeat	
timer	auto-repeat timer	
rep	auto-repeat parameters	
mt	struct input_mt holding Multitouch state	
absinfo	Absolute axes coordinate information	
key	current state of device keys/buttons	EVIOCGKEY
led	current state of device LEDs, if any	EVIOCGLED
sw	current state of device switches, if any	EVIOCGSW
open	callback for `open(2)` on device	
close	callback for `close(2)` on device	
flush	flush device events,e.g. force-feedback	
event	handler for events sent to device	

Figure figInputDev: The struct input_dev (from `<linux/input.h>`)



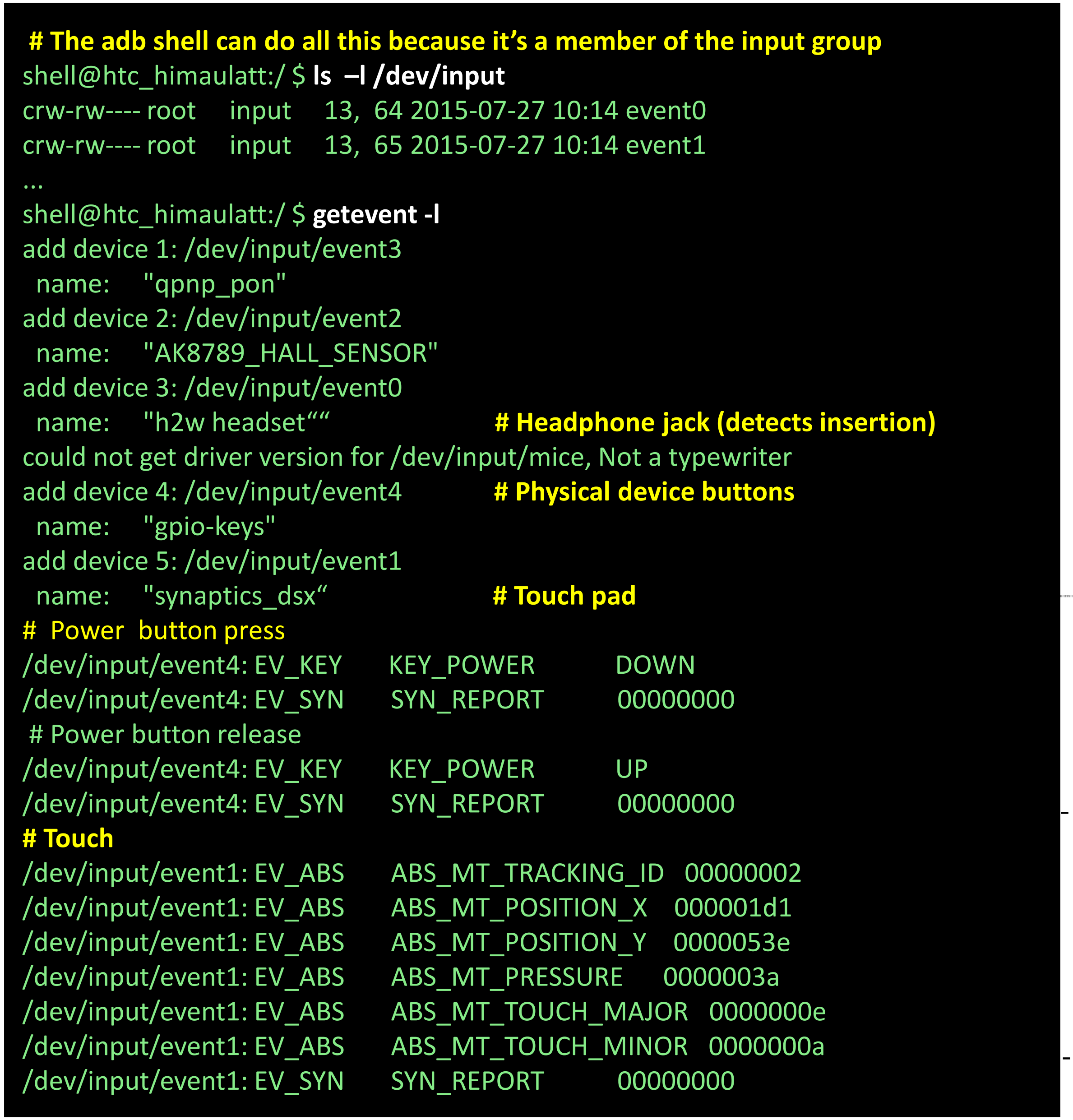
#	Event code	Specifies
0x00	`EV_SYN`	Separate/synchronize other events (e.g. `SYN_REPORT/SYN_MT_REPORT`), or report events lost (`SYN_DROPPED`)
0x01	`EV_KEY`	Key press (`KEY_*`) or touch (`BTN_TOUCH`)
0x02	`EV_REL`	Relative changes to a property. Changes relayed through `REL_[XYZ]` values.
0x03	`EV_ABS`	Absolute coordinates for an event. Values are usually `ABS_[XYZ]`, or `ABS_MT` for multi-touch
0x04	`EV_MSC`	Miscellaneous codes
0x05	`EV_SW`	Binary switches. E.g. `SW_JACK_PHYSICAL_INSERT` for headphone insertion
0x11	`EV_LED`	Used for device LEDs, if any
0x12	`EV_SND`	Used for sound devices
0x14	`EV_REP`	Used for auto-repeating events
0x15	`EV_FF`	Used for force-feedback capable devices (e.g. joysticks). An EVIOCSFF ioctl may be used to upload force feedback effects
0x16	`EV_PWR`	Reserved for power events. Largely unused
0x17	`EV_FF_STATUS`	Used for force-feedback capable devices.



CPU responds to interrupts, calls kernel, to dispatch to device driver

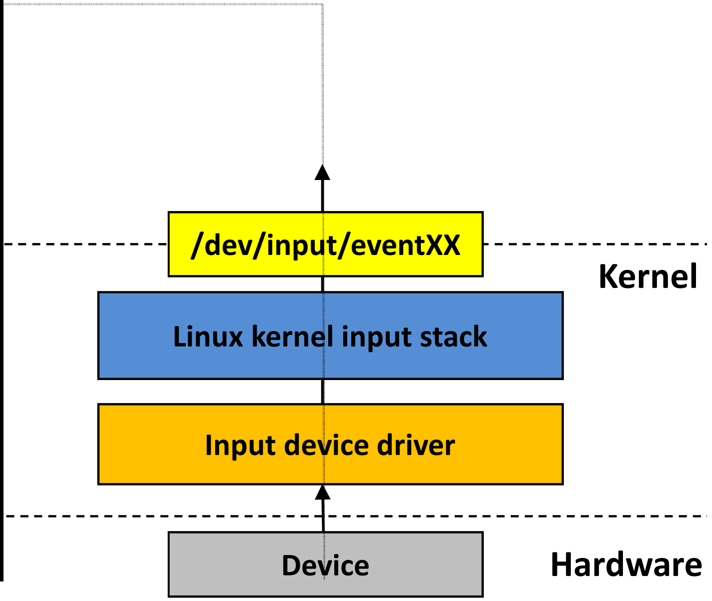
All Input starts with some type of interrupt, generated by device

The Linux Input Model



```
 # The adb shell can do all this because it's a member of the input group
shell@htc_himaulatt:/ $ ls  –l /dev/input
crw-rw---- root     input     13,  64 2015-07-27 10:14 event0
crw-rw---- root     input     13,  65 2015-07-27 10:14 event1
…
shell@htc_himaulatt:/ $ getevent -l
add device 1: /dev/input/event3
  name:     "qpnp_pon"
add device 2: /dev/input/event2
  name:     "AK8789_HALL_SENSOR"
add device 3: /dev/input/event0
  name:     "h2w headset"“                  # Headphone jack (detects insertion)
could not get driver version for /dev/input/mice, Not a typewriter
add device 4: /dev/input/event4             # Physical device buttons
  name:     "gpio-keys"
add device 5: /dev/input/event1
  name:     "synaptics_dsx“                 # Touch pad
#  Power  button press
/dev/input/event4: EV_KEY      KEY_POWER          DOWN
/dev/input/event4: EV_SYN      SYN_REPORT         00000000
 # Power button release
/dev/input/event4: EV_KEY      KEY_POWER          UP
/dev/input/event4: EV_SYN      SYN_REPORT         00000000
# Touch
/dev/input/event1: EV_ABS      ABS_MT_TRACKING_ID   00000002
/dev/input/event1: EV_ABS      ABS_MT_POSITION_X    000001d1
/dev/input/event1: EV_ABS      ABS_MT_POSITION_Y    0000053e
/dev/input/event1: EV_ABS      ABS_MT_PRESSURE      0000003a
/dev/input/event1: EV_ABS      ABS_MT_TOUCH_MAJOR   0000000e
/dev/input/event1: EV_ABS      ABS_MT_TOUCH_MINOR   0000000a
/dev/input/event1: EV_SYN      SYN_REPORT         00000000
```

The `getevent` tool reads input events



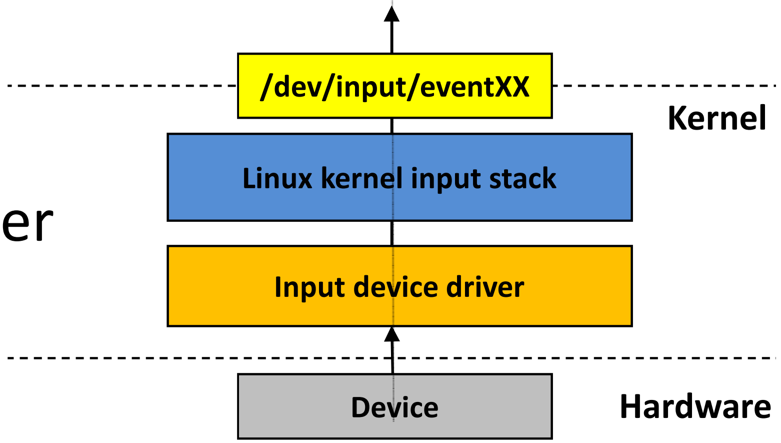
(c) 2015 Jonathan Levin - http://NewAndroidBook.com/ - Distribute Freely, but please do not plagiarize!

The Linux Input Model

```
 # Note /dev/event/input devices are also writable!
shell@htc_himaulatt:/ $ ls  –l /dev/input
crw-rw---- root     input     13,  64 2015-07-27 10:14 event0
crw-rw---- root     input     13,  65 2015-07-27 10:14 event1
...
# simulate EV_KEY KEYHOMEPAGE DOWN followed by REPORT
shell@htc_himaulatt:/ $ sendevent /dev/input/event5 1 172 1 \;
                        sendevent /dev/input5 0 0 0
# To simulate home button hold, delay the following line, simulating the UP/REPORT
shell@htc_himaulatt:/ $  sendevent /dev/input/event5 1 172 0; \
                         sendevent /dev/input5 0 0 0
```

The sendevent tool injects input events

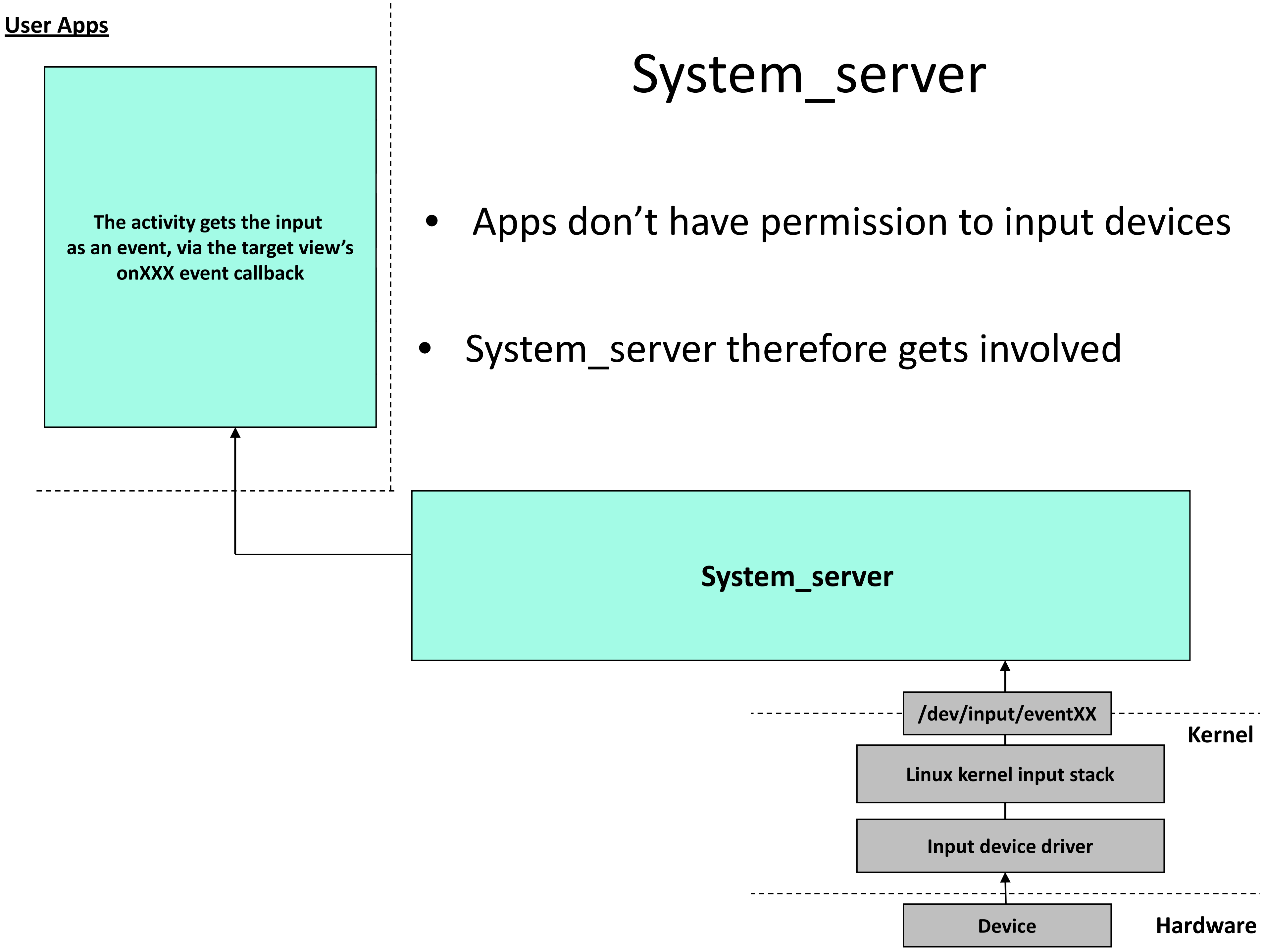
Events injected indistinguishable from driver



(c) 2015 Jonathan Levin - http://NewAndroidBook.com/ - Distribute Freely, but please do not plagiarize!

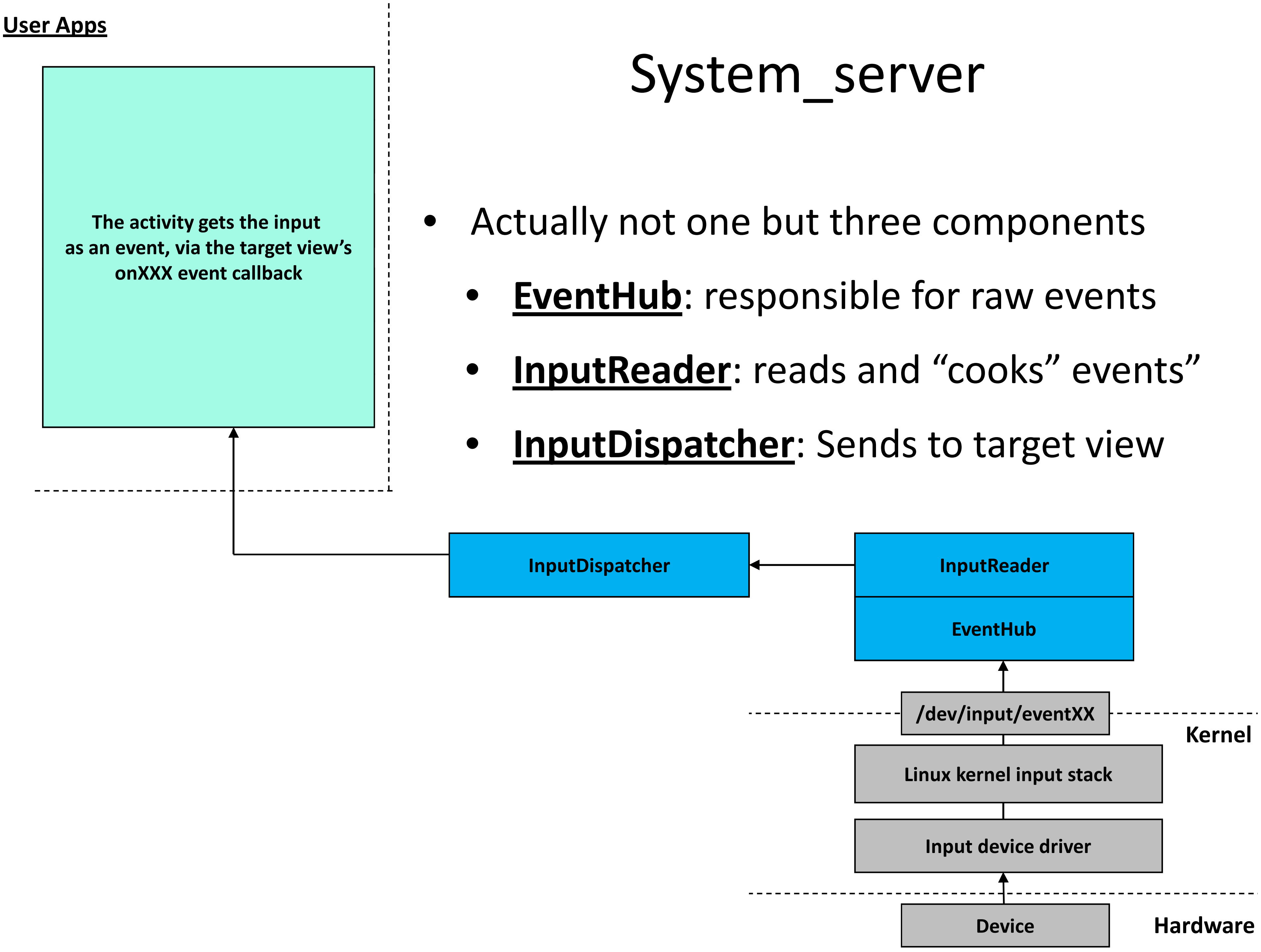
System_server

- Apps don’t have permission to input devices
- System_server therefore gets involved



System_server

- Actually not one but three components
  - **EventHub**: responsible for raw events
  - **InputReader**: reads and "cooks" events"
  - **InputDispatcher**: Sends to target view



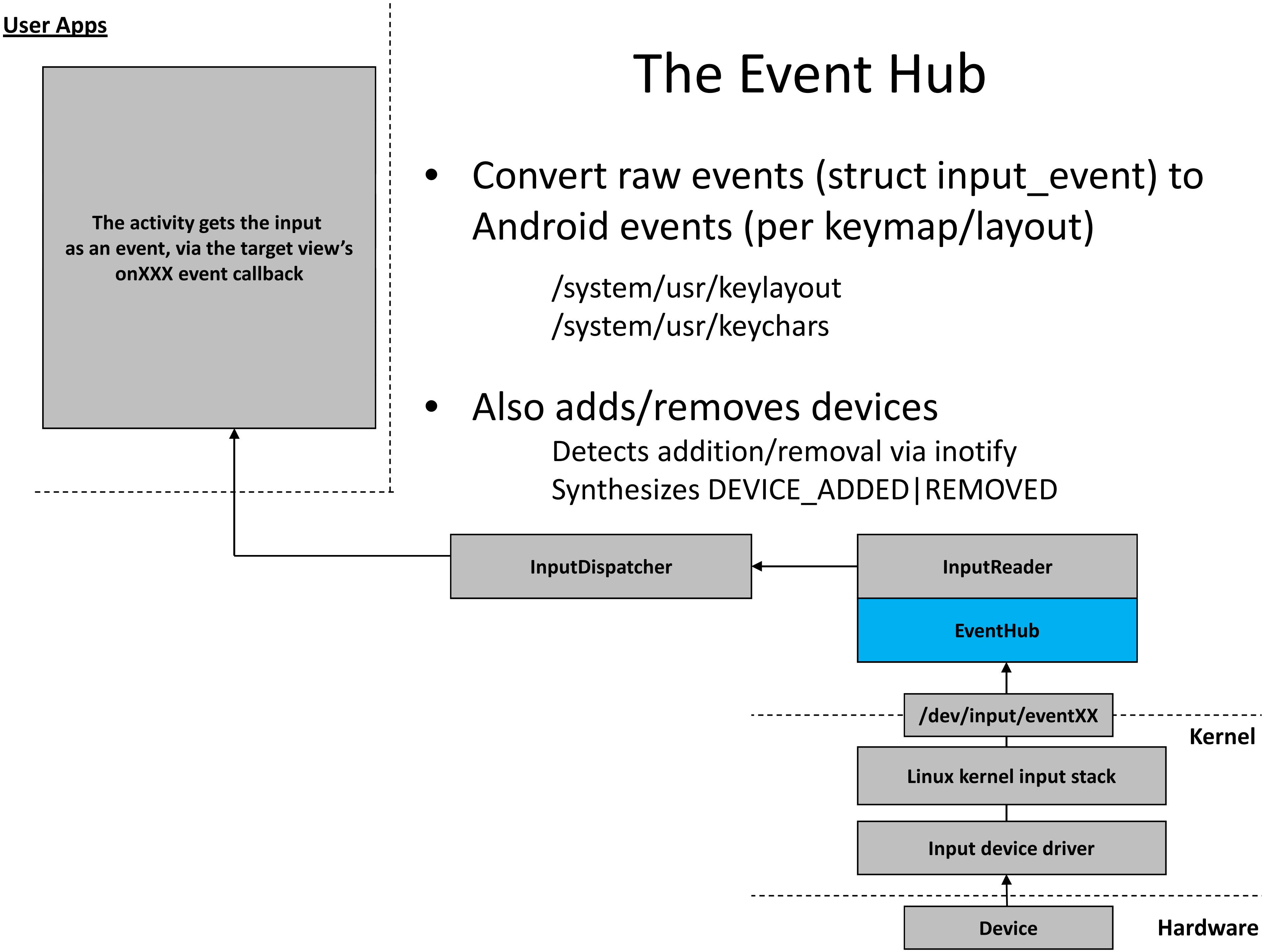
The Event Hub

- Convert raw events (struct input_event) to Android events (per keymap/layout)

  /system/usr/keylayout
  /system/usr/keychars

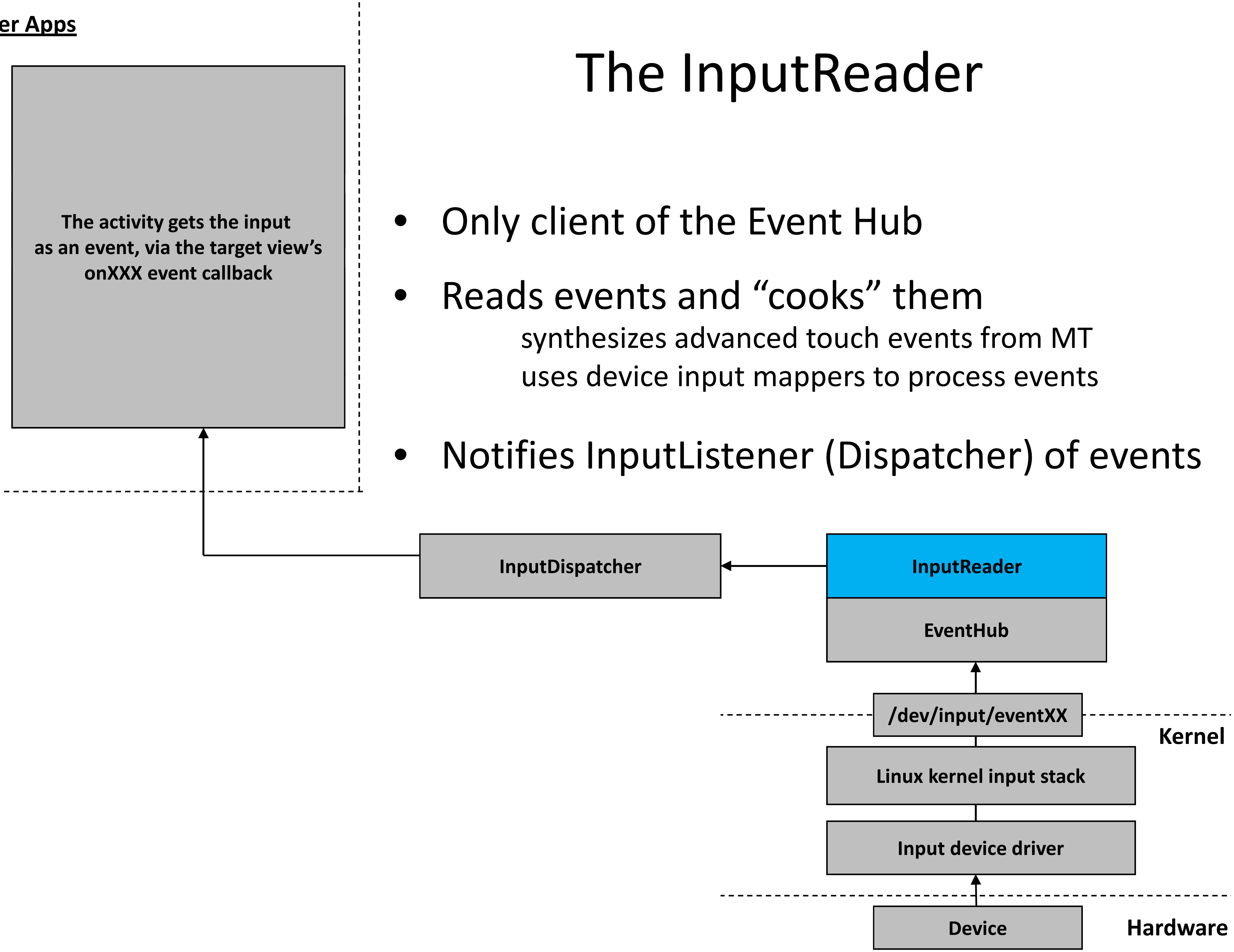
- Also adds/removes devices

  Detects addition/removal via inotify
  Synthesizes DEVICE_ADDED|REMOVED



The InputReader

- Only client of the Event Hub
- Reads events and “cooks” them
  - synthesizes advanced touch events from MT
  - uses device input mappers to process events
- Notifies InputListener (Dispatcher) of events



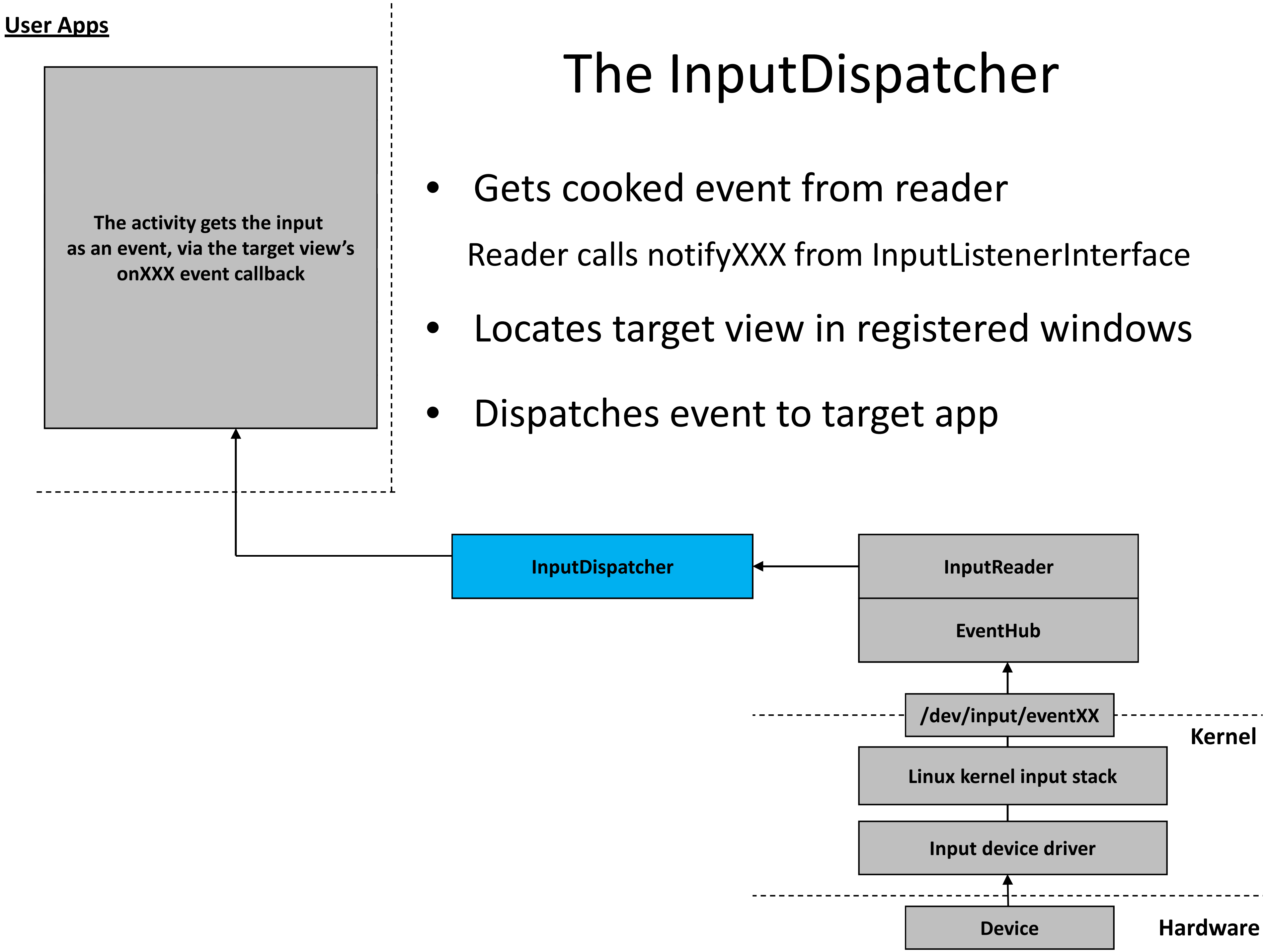
The InputDispatcher

- Gets cooked event from reader

  Reader calls notifyXXX from InputListenerInterface

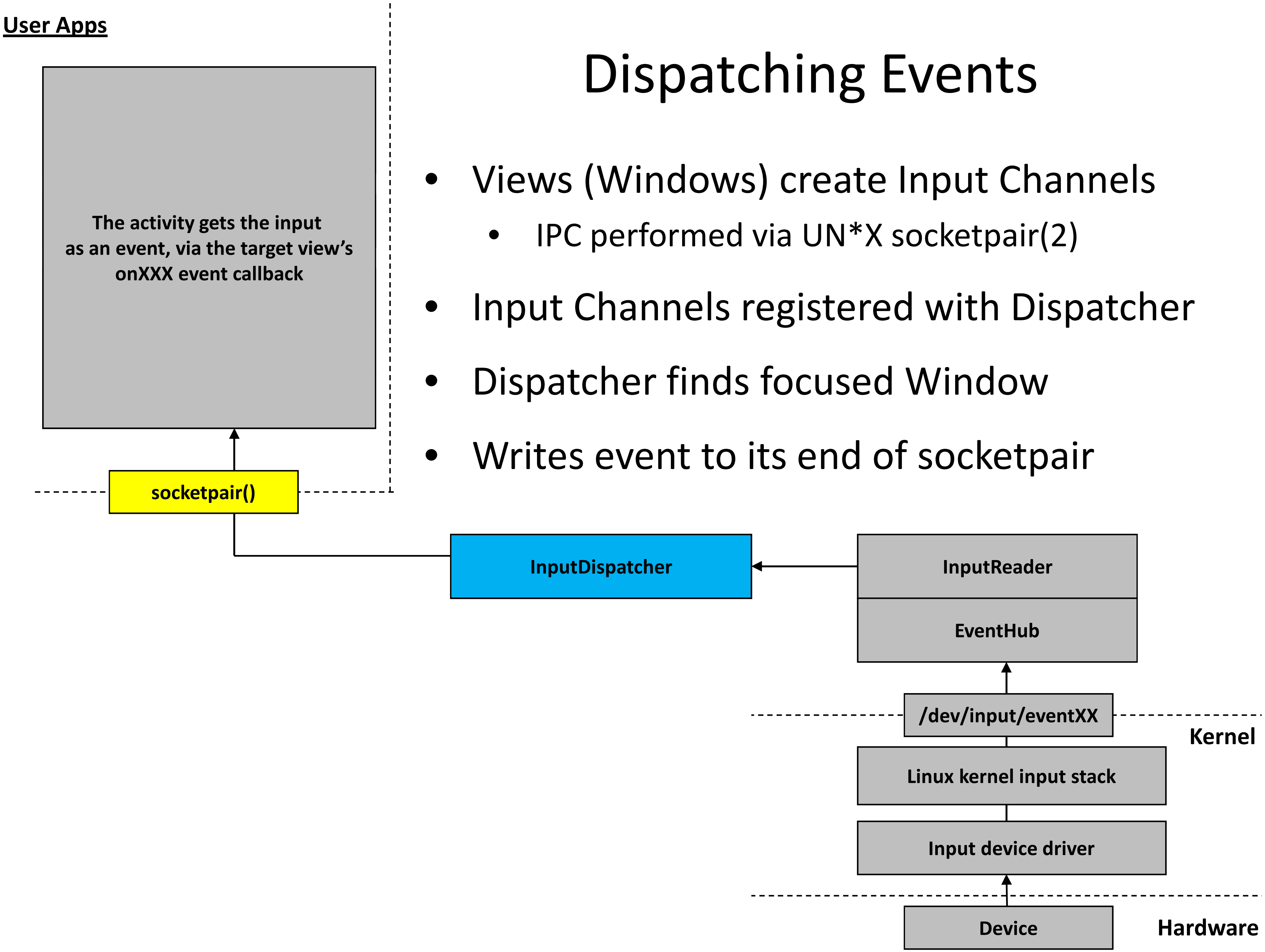
- Locates target view in registered windows

- Dispatches event to target app



Dispatching Events

- Views (Windows) create Input Channels
  - IPC performed via UN*X socketpair(2)
- Input Channels registered with Dispatcher
- Dispatcher finds focused Window
- Writes event to its end of socketpair

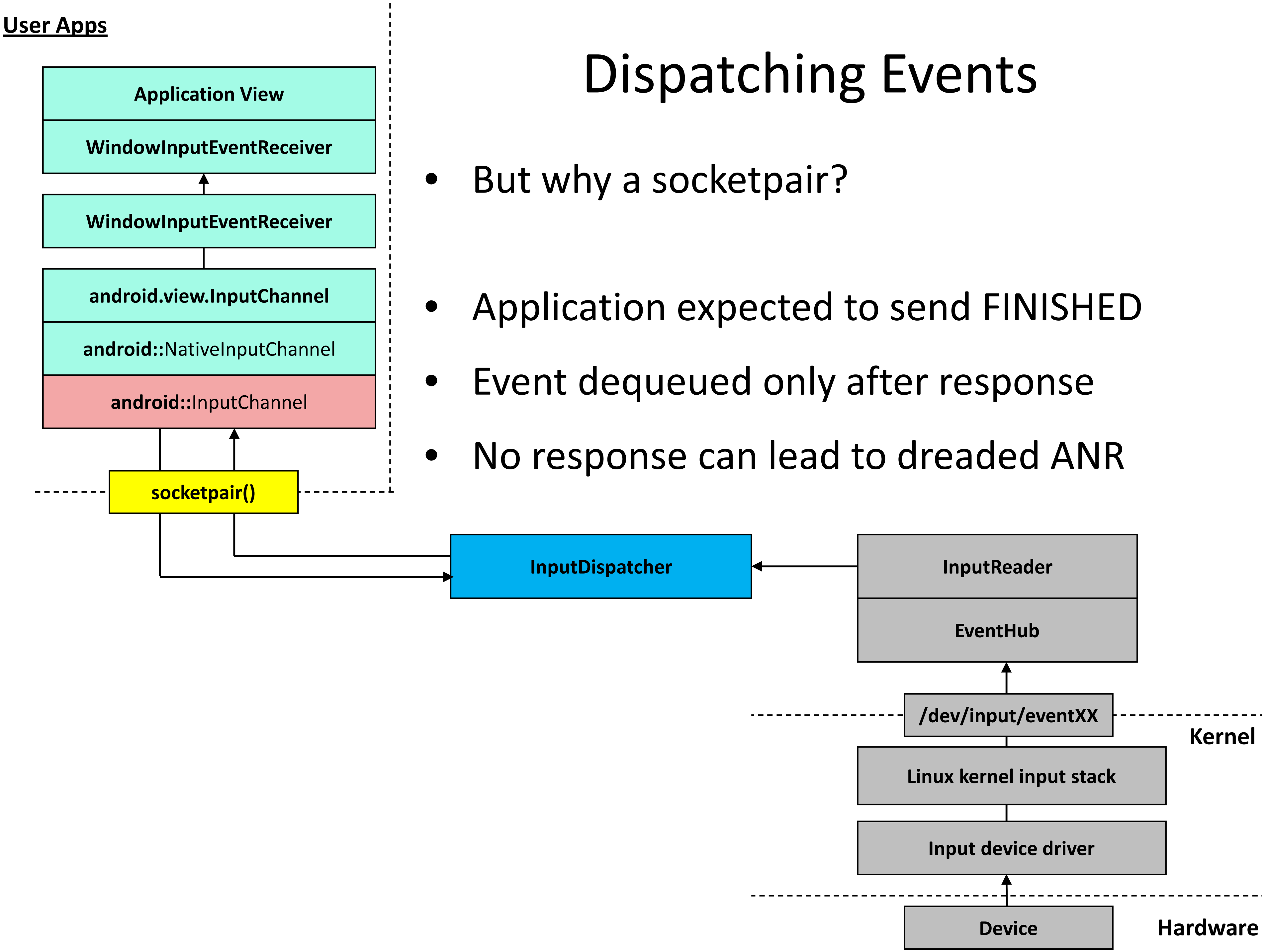


(c) 2015 Jonathan Levin - http://NewAndroidBook.com/ - Distribute Freely, but please do not plagiarize!

Dispatching Events

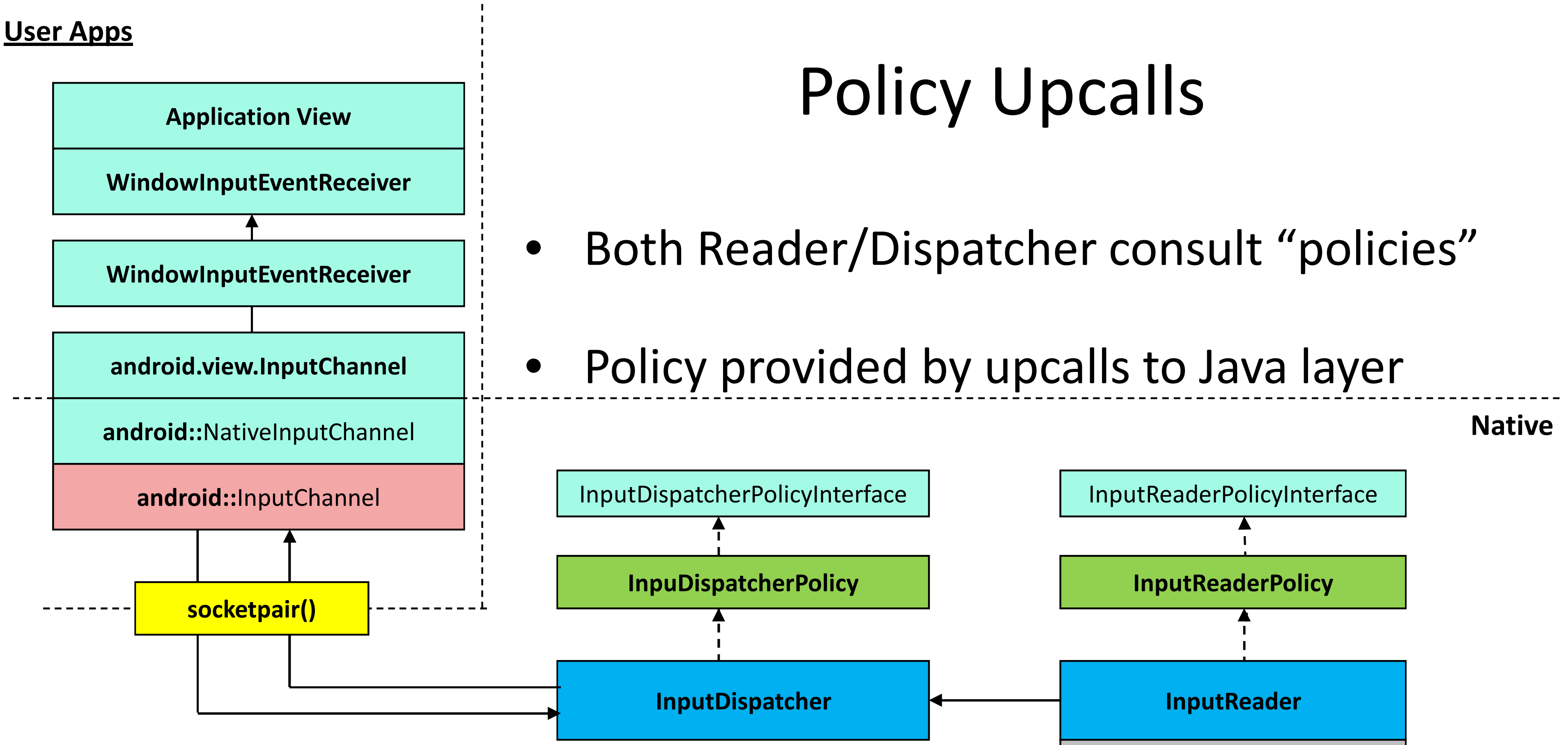
- But why a socketpair?

- Application expected to send FINISHED
- Event dequeued only after response
- No response can lead to dreaded ANR



Policy Upcalls

- Both Reader/Dispatcher consult “policies”
- Policy provided by upcalls to Java layer



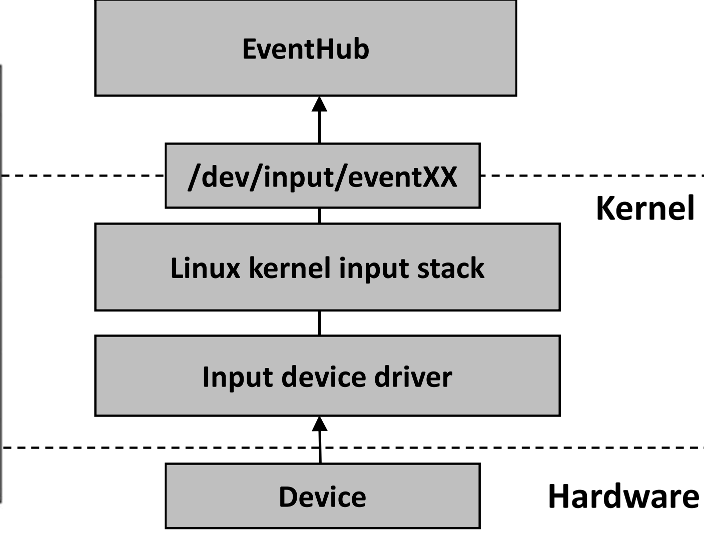
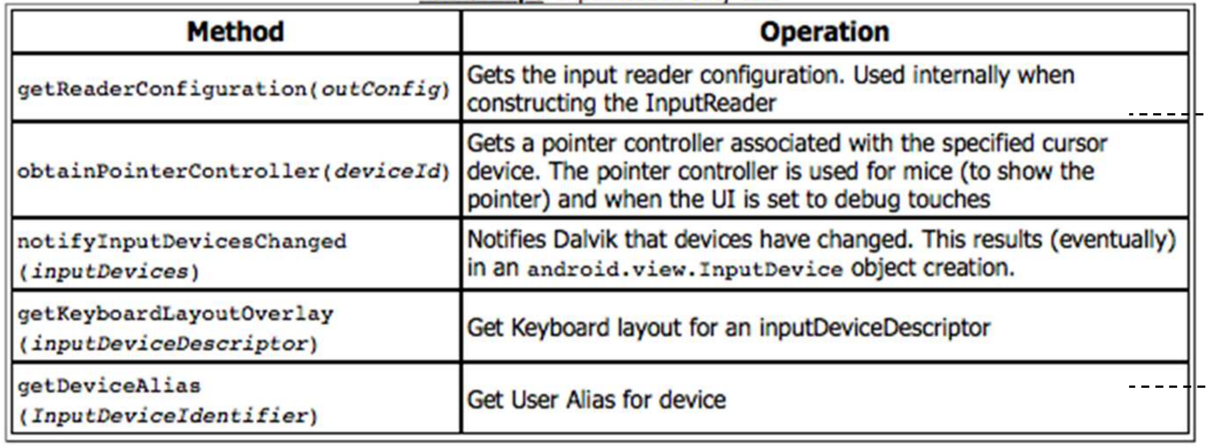


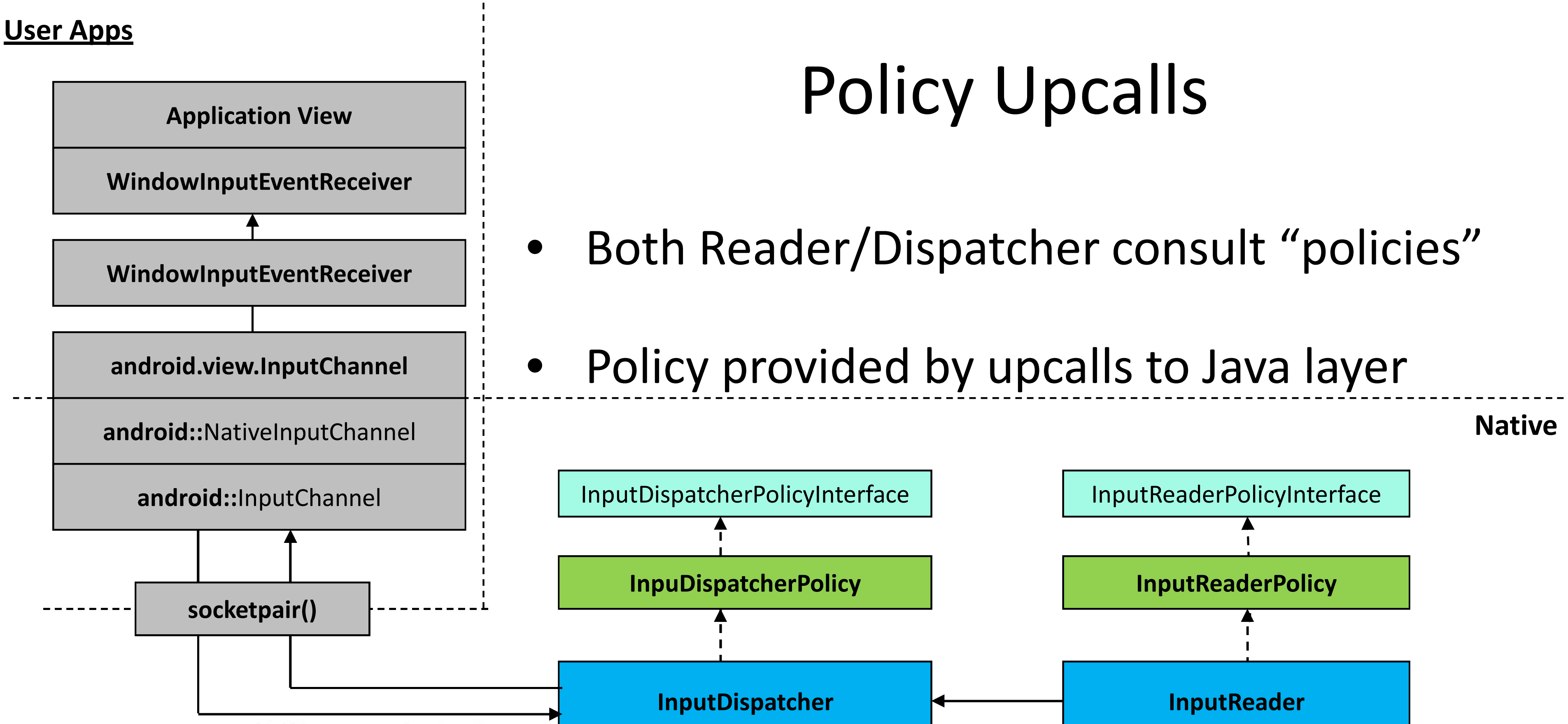
Table i-irpc: InputReaderPolicy calls



Method	Operation
`getReaderConfiguration(outConfig)`	Gets the input reader configuration. Used internally when constructing the InputReader
`obtainPointerController(deviceId)`	Gets a pointer controller associated with the specified cursor device. The pointer controller is used for mice (to show the pointer) and when the UI is set to debug touches
`notifyInputDevicesChanged (inputDevices)`	Notifies Dalvik that devices have changed. This results (eventually) in an `android.view.InputDevice` object creation.
`getKeyboardLayoutOverlay (inputDeviceDescriptor)`	Get Keyboard layout for an inputDeviceDescriptor
`getDeviceAlias (InputDeviceIdentifier)`	Get User Alias for device

Policy Upcalls

- Both Reader/Dispatcher consult “policies”
- Policy provided by upcalls to Java layer



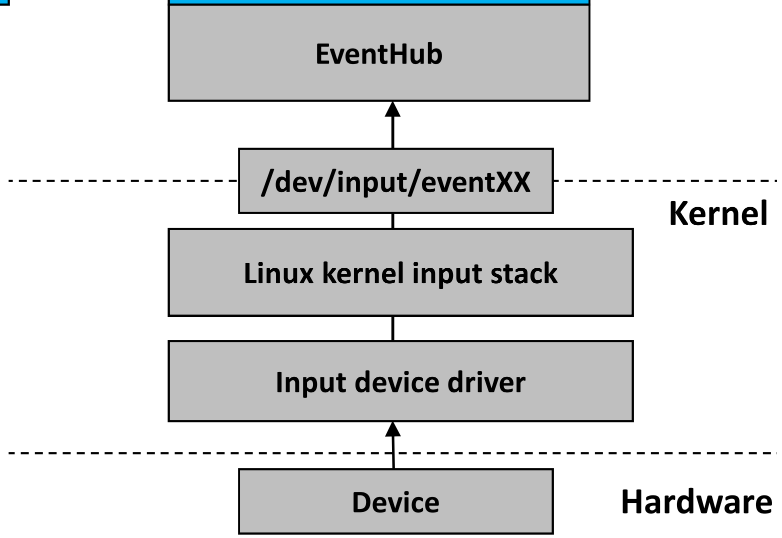
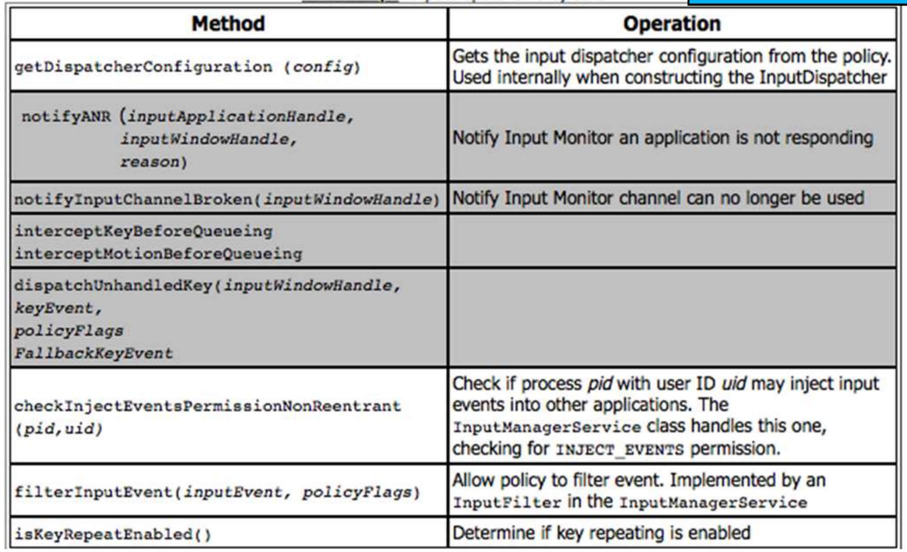


Table i-idpc: InputDispatcherPolicy calls



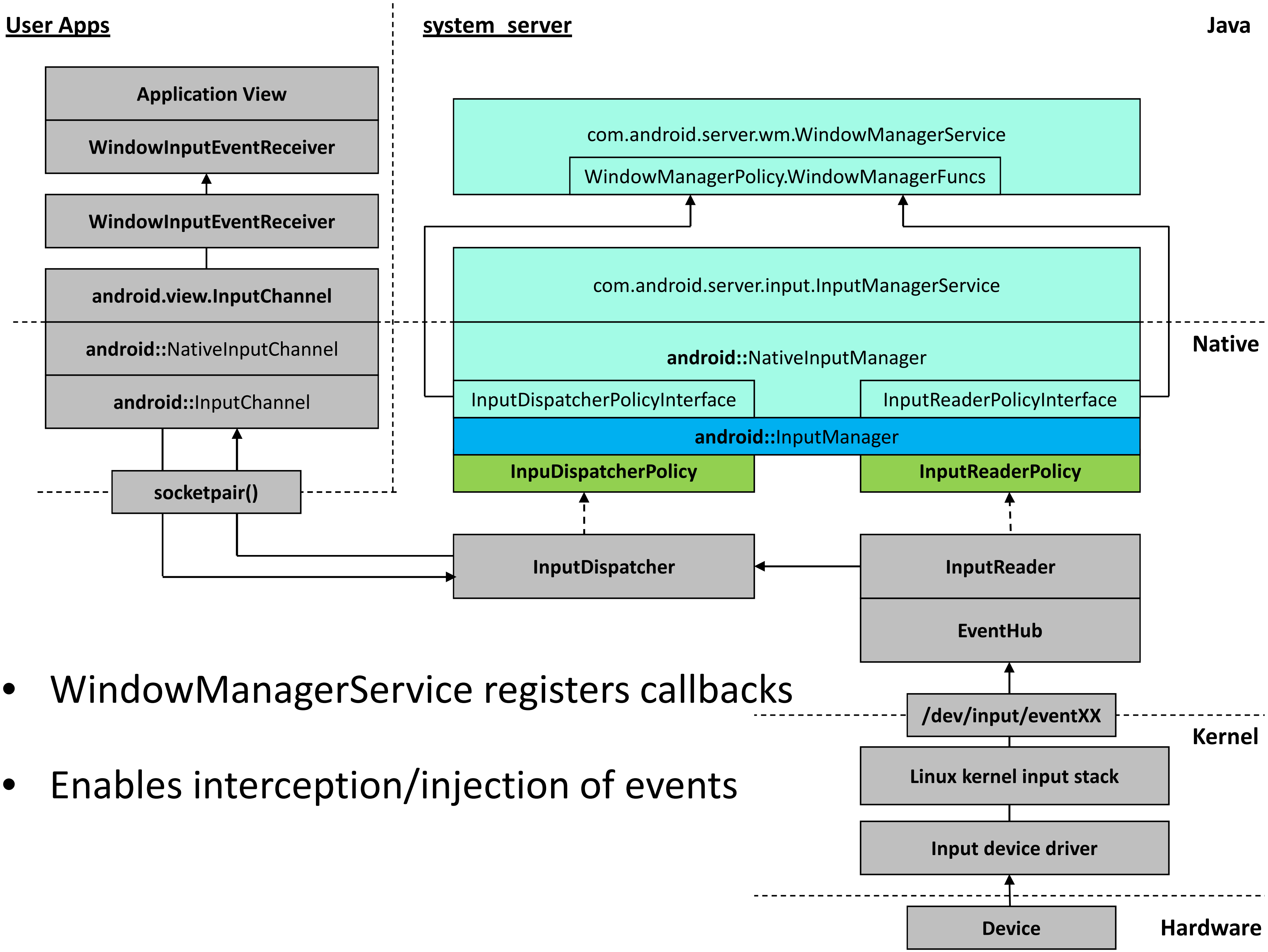
Method	Operation
`getDispatcherConfiguration (config)`	Gets the input dispatcher configuration from the policy. Used internally when constructing the InputDispatcher

`notifyANR (inputApplicationHandle,`
`inputWindowHandle,`
`reason)`
	Notify Input Monitor an application is not responding

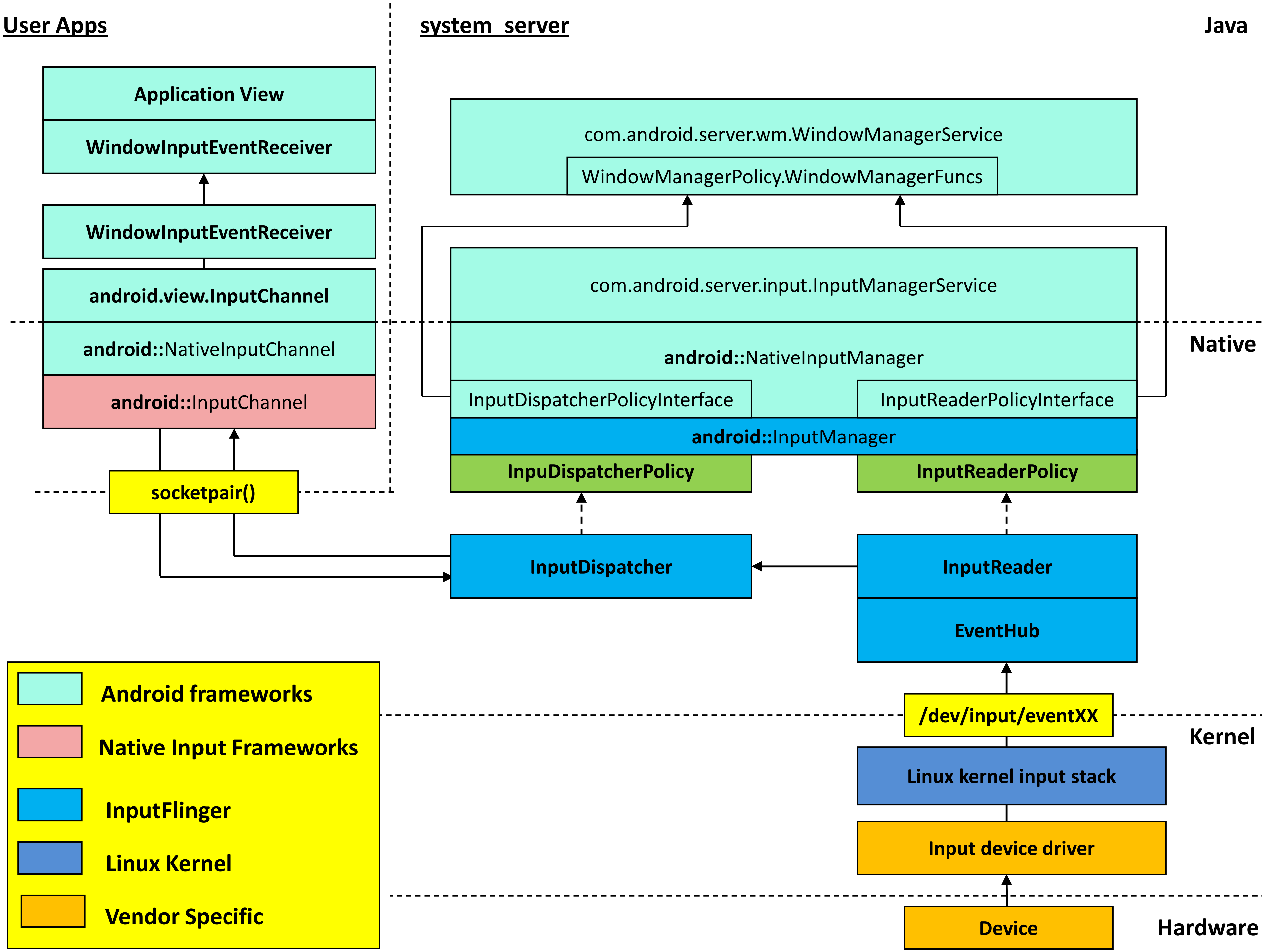
`notifyInputChannelBroken(inputWindowHandle)`	Notify Input Monitor channel can no longer be used
`interceptKeyBeforeQueueing` `interceptMotionBeforeQueueing`	

`dispatchUnhandledKey(inputWindowHandle,`
`keyEvent,`
`policyFlags`
`FallbackKeyEvent`	

`checkInjectEventsPermissionNonReentrant (pid,uid)`	Check if process *pid* with user ID *uid* may inject input events into other applications. The `InputManagerService` class handles this one, checking for `INJECT_EVENTS` permission.
`filterInputEvent(inputEvent, policyFlags)`	Allow policy to filter event. Implemented by an `InputFilter` in the `InputManagerService`
`isKeyRepeatEnabled()`	Determine if key repeating is enabled



- WindowManagerService registers callbacks
- Enables interception/injection of events



User Apps

system_server

Java

Application View

WindowInputEventReceiver

WindowInputEventReceiver

android.view.InputChannel

android:: NativeInputChannel

android:: InputChannel

socketpair()

com.android.server.wm.WindowManagerService

WindowManagerPolicy.WindowManagerFuncs

com.android.server.input.InputManagerService

Native

android:: NativeInputManager

InputDispatcherPolicyInterface

InputReaderPolicyInterface

android:: InputManager

InpuDispatcherPolicy

InputReaderPolicy

InputDispatcher

InputReader

EventHub

/dev/input/eventXX

Kernel

Linux kernel input stack

Input device driver

Device

Hardware

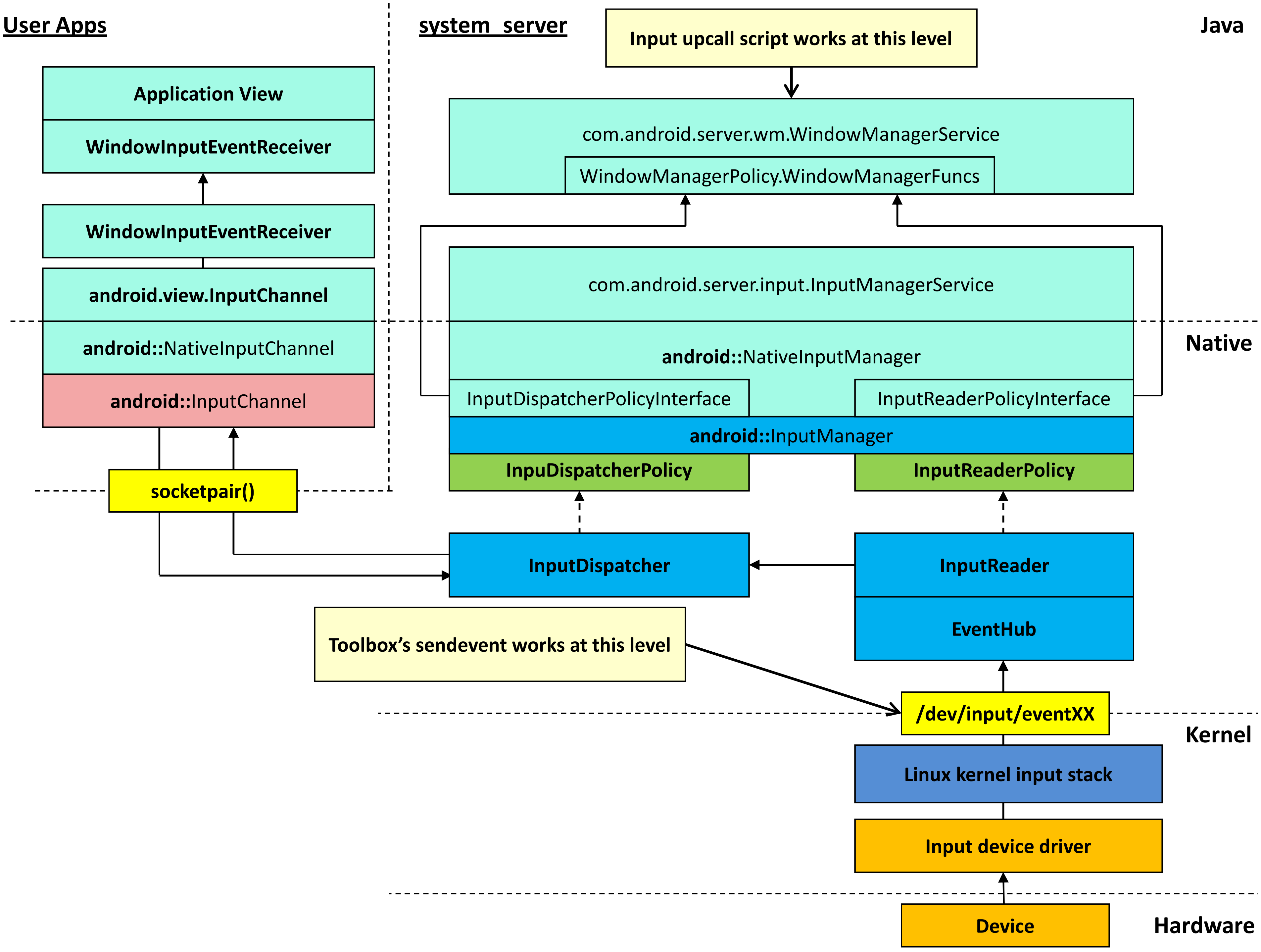
🟩 Android frameworks

🟥 Native Input Frameworks

🟦 InputFlinger

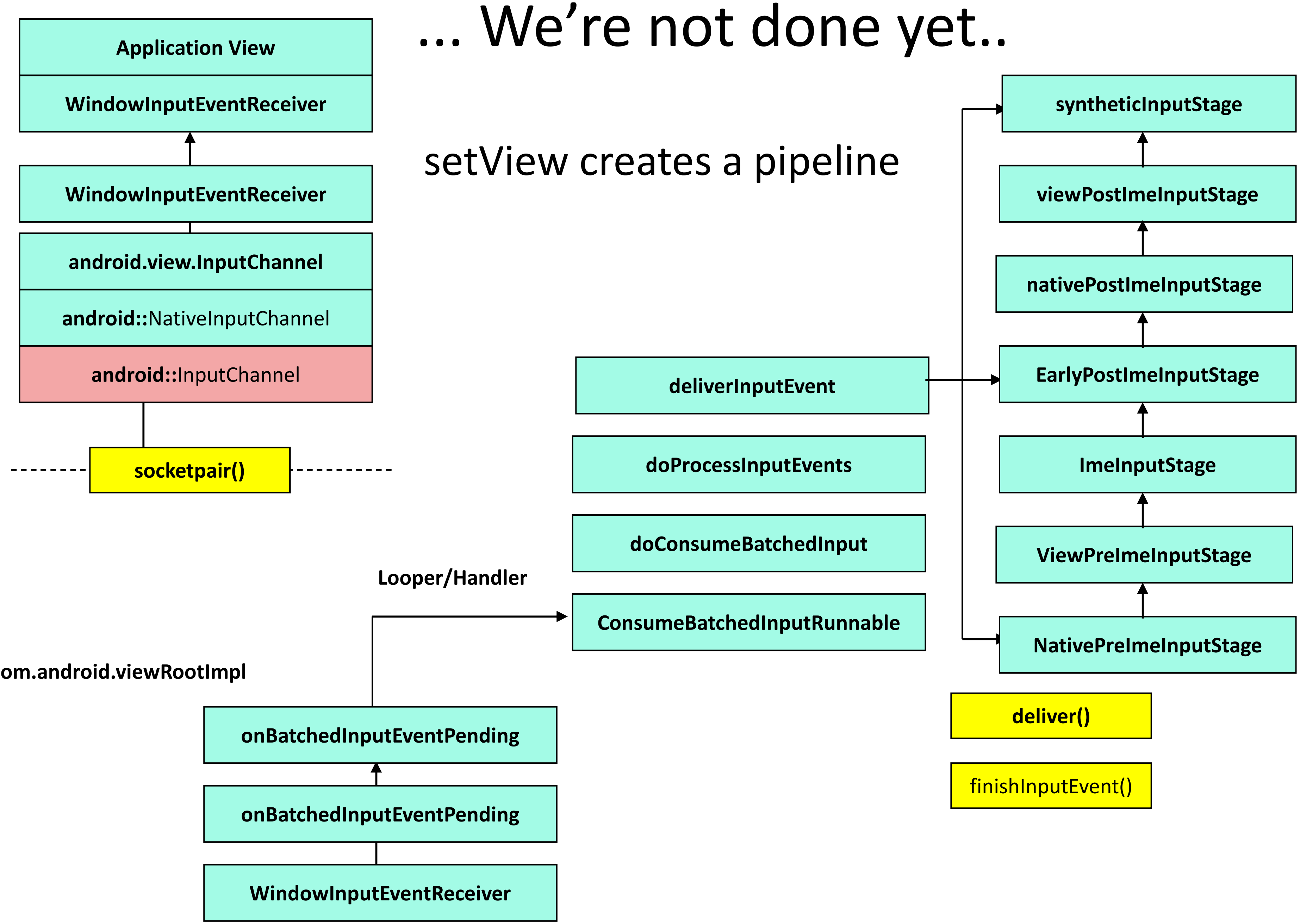
🟦 Linux Kernel

🟧 Vendor Specific


User Apps
system_server
Input upcall script works at this level
Java
Application View
WindowInputEventReceiver
com.android.server.wm.WindowManagerService
WindowManagerPolicy.WindowManagerFuncs
WindowInputEventReceiver
android.view.InputChannel
com.android.server.input.InputManagerService
android::NativeInputChannel
android::NativeInputManager
Native
android::InputChannel
InputDispatcherPolicyInterface
InputReaderPolicyInterface
android::InputManager
socketpair()
InpuDispatcherPolicy
InputReaderPolicy
InputDispatcher
InputReader
EventHub
Toolbox's sendevent works at this level
/dev/input/eventXX
Kernel
Linux kernel input stack
Input device driver
Device
Hardware

... We're not done yet..

setView creates a pipeline



Application View

WindowInputEventReceiver

WindowInputEventReceiver

android.view.InputChannel

android:: NativeInputChannel

android:: InputChannel

socketpair()

com.android.viewRootImpl

Looper/Handler

onBatchedInputEventPending

onBatchedInputEventPending

WindowInputEventReceiver

deliverInputEvent

doProcessInputEvents

doConsumeBatchedInput

ConsumeBatchedInputRunnable

syntheticInputStage

viewPostImeInputStage

nativePostImeInputStage

EarlyPostImeInputStage

ImeInputStage

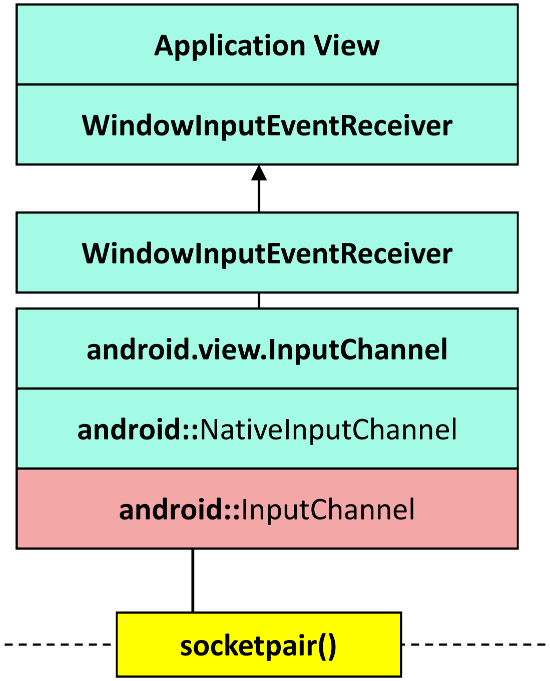
ViewPreImeInputStage

NativePreImeInputStage

deliver()

finishInputEvent()

… We're not done yet..



Application View

WindowInputEventReceiver

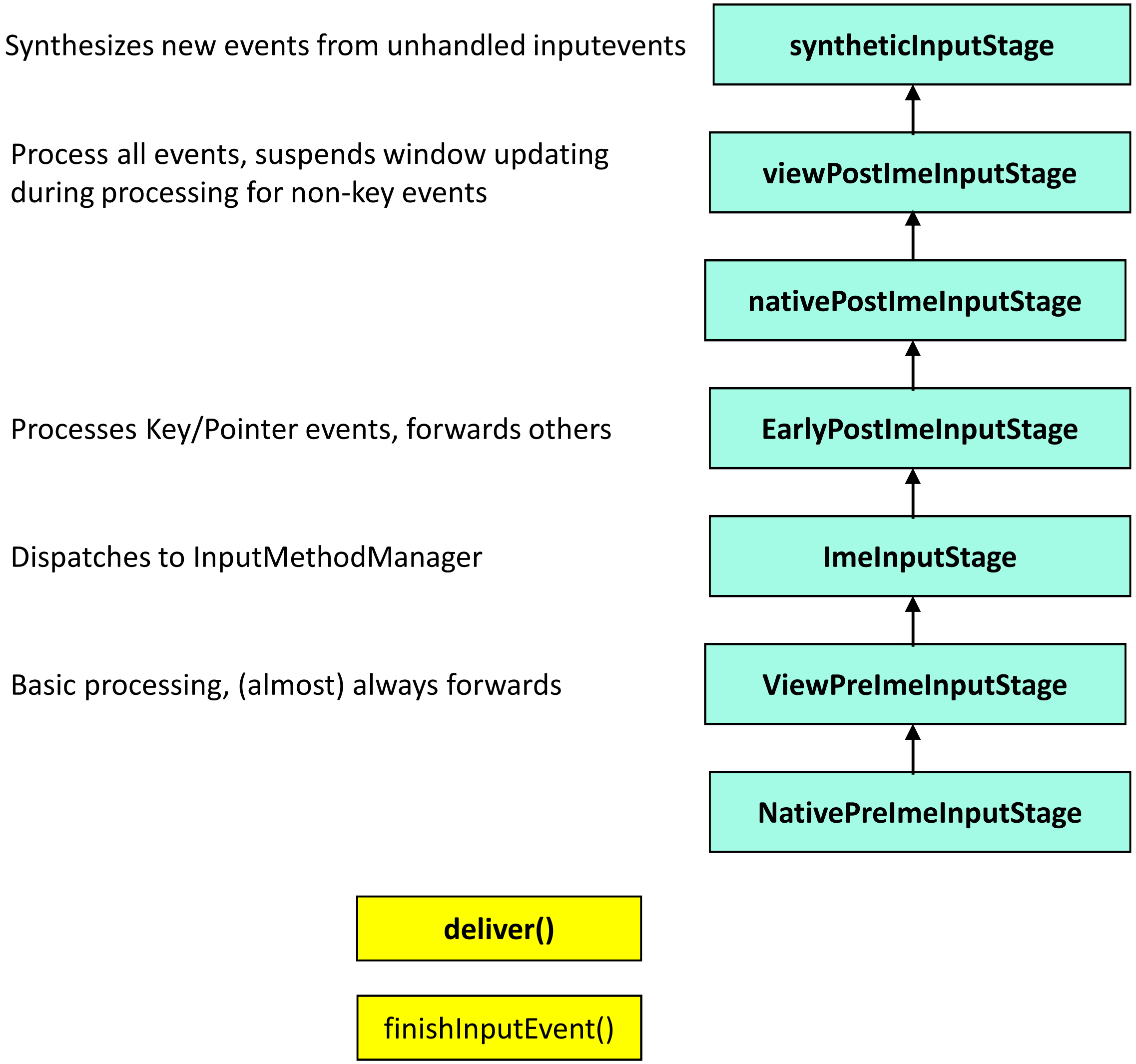
WindowInputEventReceiver

android.view.InputChannel

android:: NativeInputChannel

android:: InputChannel

socketpair()



Synthesizes new events from unhandled inputevents

syntheticInputStage

Process all events, suspends window updating during processing for non-key events

viewPostImeInputStage

nativePostImeInputStage

Processes Key/Pointer events, forwards others

EarlyPostImeInputStage

Dispatches to InputMethodManager

ImeInputStage

Basic processing, (almost) always forwards

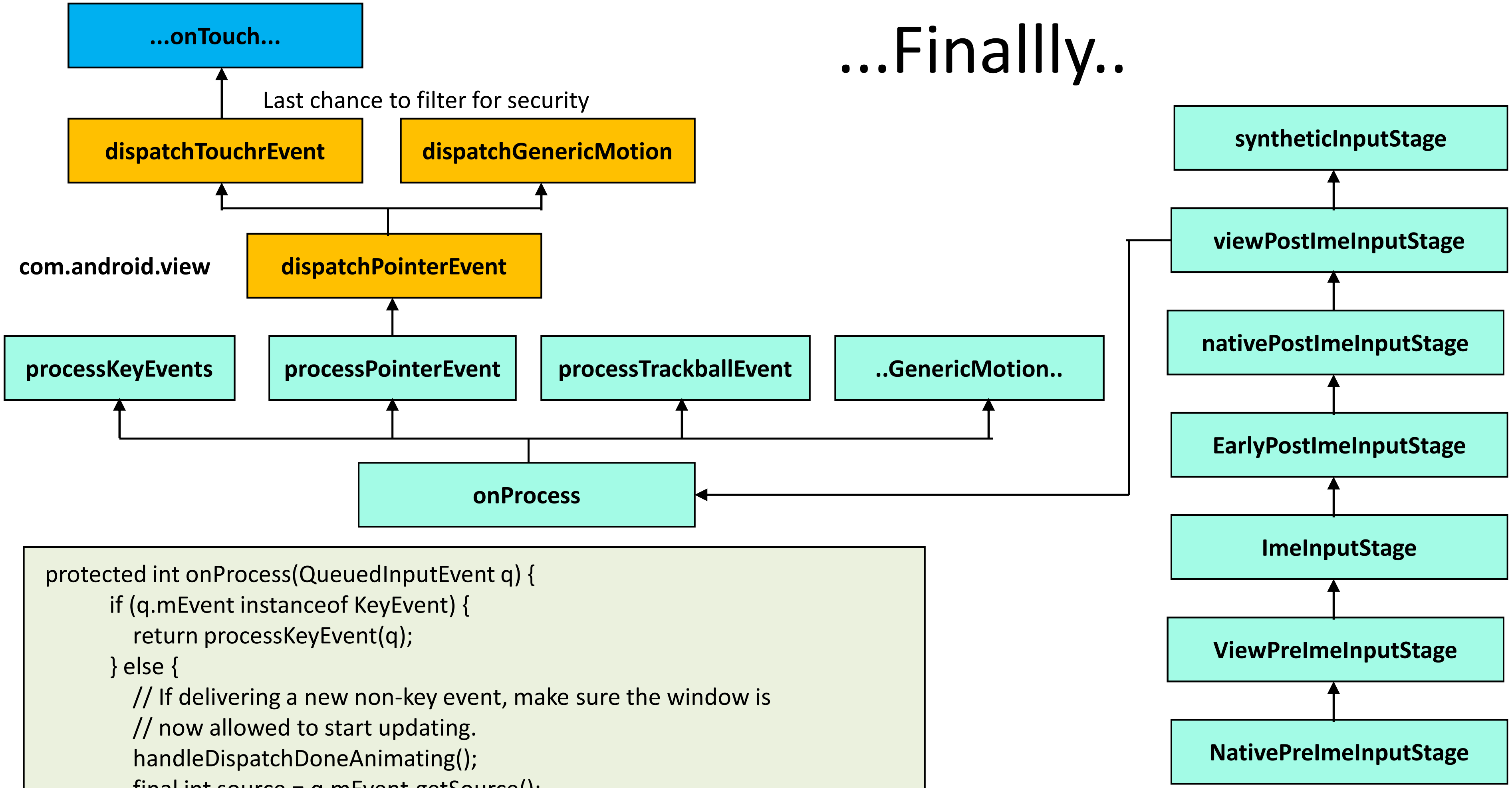
ViewPreImeInputStage

NativePreImeInputStage

deliver()

finishInputEvent()

...Finallly..

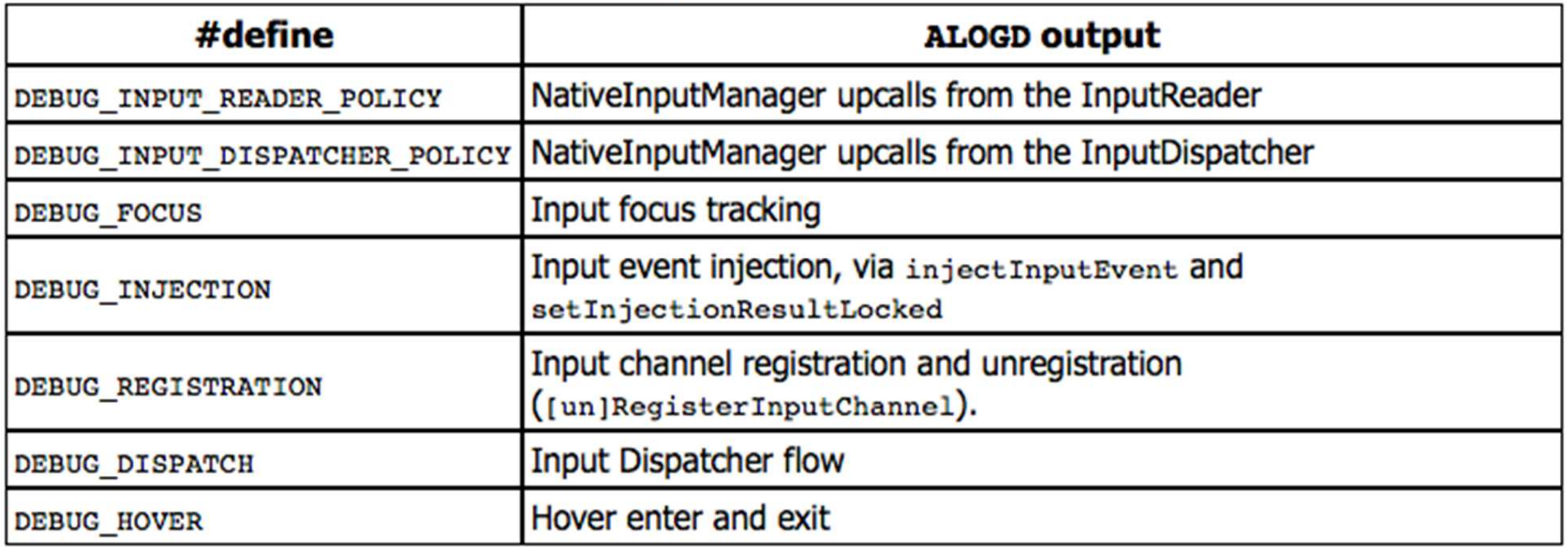


```
protected int onProcess(QueuedInputEvent q) {
        if (q.mEvent instanceof KeyEvent) {
            return processKeyEvent(q);
        } else {
            // If delivering a new non-key event, make sure the window is
            // now allowed to start updating.
            handleDispatchDoneAnimating();
            final int source = q.mEvent.getSource();
            if ((source & InputDevice.SOURCE_CLASS_POINTER) != 0) {
                return processPointerEvent(q);
            } else if ((source & InputDevice.SOURCE_CLASS_TRACKBALL) != 0) {
                return processTrackballEvent(q);
            } else {
                return processGenericMotionEvent(q);
            }
        }
    }
```

Input debugging

- If you can rebuild AOSP:

Table i-dm: Debug #defines in the Android source tree



#define	ALOGD output
`DEBUG_INPUT_READER_POLICY`	NativeInputManager upcalls from the InputReader
`DEBUG_INPUT_DISPATCHER_POLICY`	NativeInputManager upcalls from the InputDispatcher
`DEBUG_FOCUS`	Input focus tracking
`DEBUG_INJECTION`	Input event injection, via `injectInputEvent` and `setInjectionResultLocked`
`DEBUG_REGISTRATION`	Input channel registration and unregistration (`[un]RegisterInputChannel`).
`DEBUG_DISPATCH`	Input Dispatcher flow
`DEBUG_HOVER`	Hover enter and exit

- Use dumpsys input
- Use jtrace

Moral: Don’t touch your device so much!

- Have respect for your poor CPU has to go through EVERY time!

.. Find more detail in Android Internals::The Developer’s View

- More diagrams/flow tracing
- Only the bare minimum of code excerpts required
- Links/References to latest AOSP sources
- The only alternative to reading the source...

http://NewAndroidBook.com/ (preorder for Volume II available)